**Readme**

**Part 1.**

The solution of this task is divided into two sections:

1. **Explorative data analysis and model training.**

All the details including the code, data analysis, plots and model evaluations are accessible in the **EDA&ModelTraining.ipynb** file.

1. **Product-like code to generate the daily prediction and the relevant unit tests**

The source files: **prediction.py** and **prediction\_test.py**

After analyzing the data, we can summarize that:

* The data is relevant clean and there’s no missing value.
* Some features (registered and casual count) are in the reality situation not be given when predicting. The sum of these counts is the to be predicted count (‘cnt’). So that could explain why these two also highly correlate to the ‘cnt’.
* It contains both numerical and categorical features and fortunately the categorical features are already labeled.
* The to be predicted target shows some periodicity along time. That gives a clue that the trained regression model might have a good result evaluation.

Before training the models, we still need some preprocessing and feature engineering:

1. Drop the useless features
2. Chose which features as categorial (or as numerical)
3. Scale the numerical features
4. On hot encode the categorical features

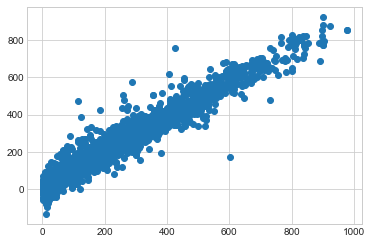
Definitely, this is a regression issue but there are lot of different models to address this problem. Considering the time and computing limitation, there are only a few to be experimented:

* The simplest linear regression model
* SVR
* XGBoost
* Lasso regression model

In order to find the best estimator, usually a grid search with cross validation should be implemented during training. Here only for SVR, it is implemented by grid search with cross validation, because even with only 18 combinations of hyperparameters it is still time consuming on PC. For this reason, the XGBoost (but using the similar addressed regression problem as reference for XGBoost) and Lasso is trained using kind of random picked hyperparameters.

Surprisingly, the XGBoost gives a quiet good result shown below (x- axis: true value; y – axis: predicted value). The XGBoost is then selected as the best estimator. (All the trained estimator are persisted to a ‘.joblib’ file.

|  |  |  |
| --- | --- | --- |
| Evaluations of Trained XGBoost | | |
| R\_2 score | mean absolute err | explained variance score |
| 0.9485 | 28.1000 | 0.9485 |



Beside the selection of models and hyperparameter, the feature engineering is also an important issue or philosophy. Because of the resource limitation, here more experiments on feature engineering would not be executed. However, we can still raise some options of this issue as future optimization:

* Take more features as categorical and then do one-hot-encoding on them (e.g. month)
* Dimension reducing (e.g. PCA)

In order to produce a daily prediction, the class *Predictor* in *prediction* module (.py) implements the predicting using a *pandas.DataFrame* instance to initialize. The unit test to it is implemented by class *PredictorTest* in *prediction\_test* module (.py).

For simplification here assumes that the input data has the same data type of each column and the same quality as the hour.csv (which means clean, no odd value, no missing value….), otherwise an extra unit should be implemented to check the quality of the input data.

**Part 2.**

According to Chen[1], the XGBoost shows a good scalability and the parallel and distributed computing is possible for it to deployed.

When the data scales to a very large set for the extremely increasing of number of instances, the training could be exhausting. This is able to be addressed through the distributed computing framework (e.g. Spark).

When the dimension of features goes really high, then deep learning might be a good choice for model, such framework like Tensorflow, Caffe…. Till now I still do not have chance to use them.

Honestly, I really enjoy solving this case even though it takes me a bit more than few hours as required. I learned a lot through it and hope you would like it.

**Reference**

[1] Chen, T., & Guestrin, C. (2016). XGBoost : Reliable Large-scale Tree Boosting System. In *Conference on Knowledge Discovery and Data Mining*. https://doi.org/10.1145/2939672.2939785